# Functional interfaces (JAVA 8)

A core concept introduced in Java 8 is that of a "functional interface**". An interface is a functional interface if it defines exactly one abstract method (‘S’ingle ‘A’bstract ‘M’ethod)**. These functional interfaces are also called **Single Abstract Method interfaces (SAM Interfaces)**. For instance, *java.lang.Runnable* is a functional interface because it only defines one abstract method:

*public abstract void run();*

Note that the "abstract" modifier is implied because the method lacks a body. It is not necessary to specify the "abstract" modifier, as this code does, in order to qualify as a functional interface.

Default methods are not abstract, so a functional interface can define as many default methods as it likes. **Functional interface cannot have more than one abstract method but it can have more than one default methods.**

If you try to add more than one abstract method to a Functional Interface will throw compilation error. For example:

*@FunctionalInterface*

*public interface WorkerInterface {*

*public void doSomeWork();*

*public void doSomeMoreWork();*

*}*

Error: Unexpected @FunctionalInterface annotation

@FunctionalInterface ^ WorkerInterface is not a functional interface multiple non-overriding abstract methods found in interface WorkerInterface 1 error

A new annotation, **@FunctionalInterface**, has been introduced. It **can be placed on an interface to declare the intention of it being a functional interface. It will cause the interface to refuse to compile unless you've managed to make it a functional interface. It's sort of like @Override in this way; it declares intention and doesn't allow you to use it incorrectly.**

To Support lambda expressions in Java 8, Functional Interfaces were introduced.

Example from JDK’s functional Interfaces. Existing interfaces made as Functional Interfaces in Java 8.

* *java.util.Comparator*
* *java.util.concurrent.Callable*
* *java.lang.Runnable*

TODO

http://eherrera.net/ocpj8-notes/04-lambda-built-in-functional-interfaces

http://www.byteslounge.com/tutorials/java-8-predicates-and-functions

http://winterbe.com/posts/2014/03/16/java-8-tutorial/

# Functional Interfaces Toolset (java.util.function)

The new package java.util.function in java 8 provides a new toolbox /rich set of functional interfaces. There are 43 of them and they all fall into the following 4 categories:

Supplier

Consumer

Predicate

Function

Predicate: Predicates represent **single argument function that returns a boolean** value.

Function: Functions also represent **a single argument function but they return a result** of an arbitrary type

# Default Methods in Interfaces (JAVA 8)

All methods of an interfaces are abstract by default. Those abstract methods don’t have implementation/definition. **From Java 8, Interface can have both abstract methods and also methods which are non-abstract** ( ie., has method definition). But only constraint is **all defined non-abstract methods in an interfaces must be ‘default’ methods** which means while defining methods ‘default’ keyword should be used before the method signature.

Why default methods?

Default method should be used for backward compatibility which means whenever we **want to add additional functionality in an existing legacy interface**, we can use default **methods without breaking any existing implementer classes** but same cannot be achieved with abstract classes which forces sub classes to adopt the change.

Strings and StringJoiner (JAVA 8)